Bootstrap and Troubleshooting

(slides are based on Prof. Dr. Jian-Jia Chen and http://www.freertos.org)

Anas Toma

LS 12, TU Dortmund

January 24, 2019

Bootstrapping

- The entire boot process of a system, from application of power to performing its intended function
- Usually just called booting
- Procedure in PC (e.g., Linux)
  - Power on
  - BIOS gets control
  - BIOS initializes some hardware
  - BIOS loads bootloader
  - Bootloader loads operating system kernel
  - Kernel probes hardware
  - Kernel finds and mounts root filesystem
  - Kernel runs init
  - Init gets userspace up and running

How to Boot Embedded Systems?

- No BIOS; initial control passes to bootloader
- Limited options for storage
- Hardware usually remains the same
- Constraints on storage and memory: only need minimal root filesystem and userspace

Outline

Bootstrapping

Troubleshooting

Outline

- Susceptible areas
  - Memory
  - Timeliness
  - Concurrency issues
- Tools and techniques
  - Code coverage tools
  - Unit tests
  - JTAG debugging

Memory issues - Stack

- Every thread in FreeRTOS has individual stack
- Stack requirement is often unpredictable
- Most common cause of spurious failures
- Particular high stack usage with library functions
  - printf, sprintf: better to write your own lightweight variant if the whole functionality is not required
- API Help:
  - uxTaskGetStackHighWaterMark(...): for testing phase
  - configCHECK_FOR_STACK_OVERFLOW: for runtime

Memory issues - Stack (2)

configCHECK_FOR_STACK_OVERFLOW

- Calls a hooked function if a stack overflow is detected
  - The application must provide it
  - Prototype: void vApplicationStackOverflowHook(TaskHandle_t xTask, signed char *pcTaskName);
- Checks made during the context switch
  - Makes context switch slower
- Three options possible
  - =0 : no checks
  - =1 : checks the current value of stack pointer
    - Fast but does not guarantee to find all stack overflows
  - =2 : checks the value of guard bytes between the stack spaces of different threads
    - In addition to =1 above

Memory Issues - Memory Allocation

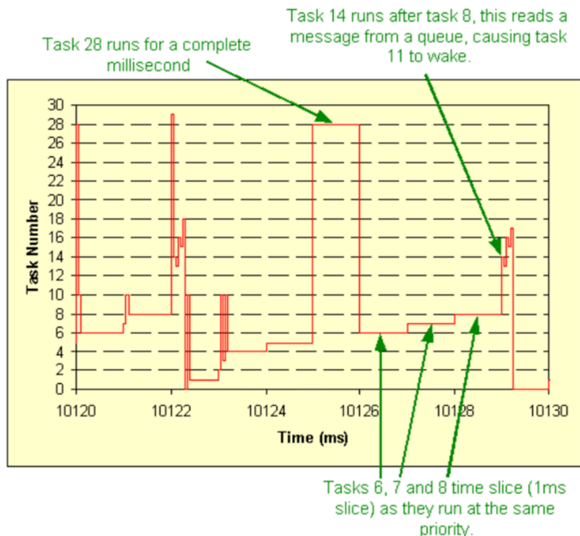
- `malloc()` and `free()`: often not a good idea for embedded systems
- Dynamic memory allocation seldom used on safety critical parts of embedded systems due to:
  - **Sufficiency:** will a critical memory demand always be met
  - **Fragmentation:** what if all the available chunks are smaller than required chunks
  - **Garbage collection:** can this process be time-bounded
  - **Timeliness:** What is the upper bound on timeliness of fulfilling a memory request
- Static memory allocation
  - might be wasteful
  - inflexible
  - but less error prone

Timeliness

Most important concern in RT embedded systems.

- Offline: verification using static analysis
  - WCET tools: aiT, Chronos et al.
  - Scheduling policy analysis using Real-time calculus
- Online: In system verification using tracing
- FreeRTOS allows tracing:
  - **Context switch** time, reason
  - **Queue** create, send, receive, peek, delete
  - **Mutex** create, give, failed
  - **Semaphore** create, give, failed
  - **Task** create, delay, resume, priority set, delete
- More can also be added
- Heisenberg bugs: Instrumenting the code changes the behavior of the code

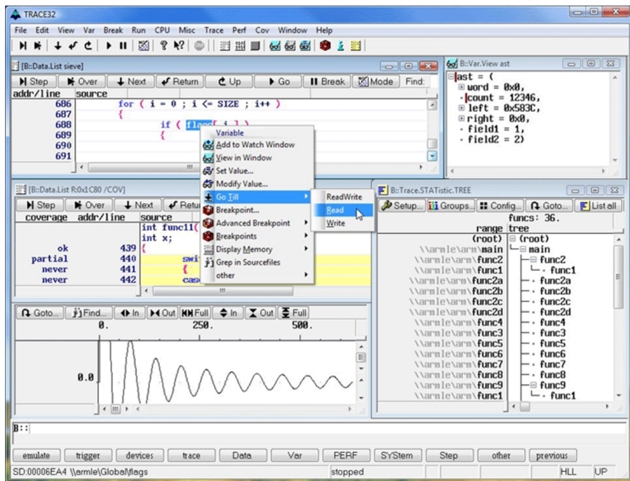
Timeliness - Tracing Utility from FreeRTOS



Concurrency issues

- Race conditions
  - The output is dependent on the sequencing or timing of the input
  - Resource Access should be carefully planned
  - Priorities inverted
- Interrupt priorites can cause problems
  - Nested interrupts can result in:
    - deadlocks
    - Interrupt misses

Trace32 - A nonfree FreeRTOS tool



Unit Tests

- "Code a little, test a little" scheme
- Test the smallest possible units of code (function) in isolation from the complete application
- Saves time in integration
- Tools available for C
  - CUnit
  - Check

JTAG Debugging

- stands for Joint test action group
- Developed in 1980s by a consortia of over 200 member companies
- Main idea: have the test facilities / test points into the chips
- Standardized protocol using 5 pins
- The hardware implementation is normally available on chip, and can be accessed through serial/USB
- Allows single stepping while being in circuit, memory/register content reading + editing

JTAG via GDB and Eclipse

Debug - Hello_World/hello_world.c - Eclipse

File Edit Source Refactor Navigate Search Run Project Window Help

Debug

Step Over (F6)

Hello_World Automatic [C/C++ Remote Application]
gdbserver debugger (8/11/11 1:52 PM) (Suspended)
Thread [1] (Suspended)
1 main() hello_world.c:5 0x00008394
Remote Shell
arm-linux-gdb (8/11/11 1:52 PM)
/home/user/workspace/Hello_World/hello_world (8/11/11 1:52 PM)

(x)= Variables | Breakpoints | Registers | Modules

Name	Value
(x)= i	2

hello_world.c | Makefile

```c
#include <stdio.h>

int main (void) {
    int i = 1;
    for(i = 0; i < 100; i++) {
        printf("Hello World.\n");
    }
    return 0;
}
```

Outline | Disassembly

Enter location here

```
0000838c:   ldr r0, [pc, #44]      ; 0x83c0 <main+88>
00008390:   bl 0x8294 <puts>
5             for(i = 0; i < 100; i++) {
00008394:   ldr r3, [r11, #-16]
00008398:   add r3, r3, #1   ; 0x1
0000839c:   str r3, [r11, #-16]
000083a0:   ldr r3, [r11, #-16]
000083a4:   cmp r3, #99      ; 0x63
000083a8:   ble 0x838c <main+36>
8             return 0;
```

Tasks | Problems | Remote Systems | Terminals | Console | Memory

Hello_World Automatic [C/C++ Remote Application] Remote Shell

```
Remote debugging from host 172.21.10.65
Hello World.
Hello World.
Hello World.
```